

Seminarul 1

Clase. Encapsulare. Constructori. Destructori

1 Clase

O clasă reprezintă extensia a structurii de date, unde, pe lângă membrii care păstrează starea/ datele (*proprietăți*), există și funcții ca membrii (*metode*).

```
1  class <nume_clasa> {
2      <specificator_access>:
3          /* definitii proprietati & metode */
4      <specificator_access>:
5          /* definitii proprietati & metode */
6
7      <numeClasa> (); // constructor
8      ~<nume_clasa> (); // destructor
9  } <list_obiecte>;
```

Exemplu:

```
1  class Point {
2      int x = 3, y = -5;
3
4      public:
5          void show () {
6              std::cout << "(" << x << ", " << y << ")";
7          }
8  } p;
```

Se pot declara clase folosind și **struct** :

```
1  struct <nume_structura> {
2      <specificator_access>:
3          /* definitii proprietati & metode */
4      <specificator_access>:
5          /* definitii proprietati & metode */
6
7      <nume_structura> (); // constructor
8      ~<nume_structura> (); // destructor
9  } <list_obiecte>;
```

Exemplu:

```
1  struct Point {
2      int x = 3, y = -5;
3  } p;
```

2 Specificatori de access

Un concept de baza in POO este **encapsularea informatiei**: ascunderea informatiei si oferirea unei interfete prin care se pot executa un set de operatii pe acea informatie.

Q: De ce encapsulare? A: Pentru ca, de cele mai multe ori, operarea directa pe datele unei structuri poate duce la rezultate neprevazute atunci cand modificarea datelor presupune anumiti algoritmi. Prin encapsulare se poate forta ca orice operatie defnita pe un set de date sa fie facute doar prin mecanismele definite, care modifica datele intr-o maniera consistenta si sigura.

In C++ avem 3 specificatori:

- **private** - access permis doar in interiorul clasei
- **protected** - access permis in interiorul clasei si in clasele derivate
- **public** - access permis peste tot.

Este recomandat ca toate proprietatile unei clase sa fie declarate cu specificatorul **private** sau **protected**

In cazul claselor declarate folosind cuvantul cheie **class**, specificatorul de access default este **private**. In cazul claselor declarate folosind cuvantul cheie **struct**, specificatorul default de access este **public**.

3 Constructori

Un constructor reprezinta "reteta" dupa care se creaza obiectul. Constructorul spune compilatorului cum anume trebuie construit un obiect dintr-o clasa. Constructorul se declara similar cu metodele, cu urmatoarele precizari: numele este identic cu numele clasei si nu exista tip de retur:

```
1  class Point {  
2      int x = 3, y = -5;  
3  
4      public:  
5          Point (int a, int b) {  
6              x = a;  
7              y = b;  
8          }  
9          void show () {  
10             std::cout << "(" << x << ", " << y << ")";  
11         }  
12     };
```

Daca o clasa nu are definit un constructor, compilatorul va atribui un constructor implicit pentru acea clasa.

Exista mai multe tipuri:

- constructor fara parameterii
- constructor cu parameterii
- constructor de copiere

Constructorul cu parametrii si constructorul fara parametrii sunt apelati la declararea obiectului. Aceste doua tipuri de constructor spun cum anume se creaza un obiect atunci cand nu avem informatii despre obiect sau cand avem informatii despre obiect.

Exemplu:

```
1
2 class Point {
3     int x = 3, y = -5;
4
5     public:
6         Point () {
7             x = 0;
8             y = 0;
9         }
10        Point (int a, int b) {
11            x = a;
12            y = b;
13        }
14    };
15
16 int main () {
17     Point p; // se apeleaza constructorul fara parameterii
18     Point o(1, -10); // se apeleaza constructorul cu parameterii
19     return 0;
20 }
```

Constructorul de copiere este apelat la apelul unei functii care primeste ca parametru un obiect de tipul clasei. Scopul este sa se creeze o copie a obiectului curent, care sa fie folosita in apelul functiei, a.i. dupa terminarea apelului obiectul curent sa ramana nemodificat.

```
1 class Point {
2     int x = 3, y = -5;
3
4     public:
5         Point (Point& p) {
6             x = p.x;
7             y = p.y;
8         }
9    };
10
11 void foo (Point o) {
12     /* corpul functiei */
13 }
14
15 int main () {
16     Point p;
17     Point o(p); // se apeleaza explicit constructorul de copiere
18     f(p); // se apeleaza implicit constructorul de copiere
19     return 0;
20 }
```

Este absolut necesar ca parameterul pasat catre constructorul de copiere sa fie pasat prin referinta. Altfel se ajunge la un lant infinit de apeluri recursive.

4 Destructori

Destructorul este opusul constructorului. Scopul unui destructor este sa spuna compilatorului cum anume se distruge obiectul. Destructorul este apelat automat de compilator atunci cand durata de viata a unui obiect ajunge la sfarsit.

La fel ca in cazul constructorilor, destructori se declara ca orice alta metoda, avand numele clasei cu `~` in fata si niciun tip de retur

```
1  class Point {
2      int x, y;
3
4      public:
5          Point () {
6              x = 3;
7              y = 44;
8          }
9
10         ~Point () {
11             std::cout << "Obiect distrus";
12         }
13     };
14
15     int main () {
16         Point p;
17         return 0;
18     } // se apeleaza destructorul
19
```

Daca nu este declarat un destructor compilatorul creaza un destructor implicit.

5 Exerciții

1. Implementati clasa **Stiva** avand urmatoarele functionalitati:

- memorie alocata dinamic
- constructor (de toate tipurile) si destructor
- metoda de adaugare element in Stiva
- metoda de eliminare element din Stiva care returneaza elementul eliminat;

2. Implementati clasa **Vector** avand urmatoarele functionalitati:

- memorie alocata dinamic
- constructor (de toate tipurile) si destructor
- metoda de adaugare element la sfarsitul vectorului
- metoda de eliminare element de la sfarsitul vectorului
- metoda ce intoarce un element de pe pozitia `i` primita ca parameteru
- metoda ce intoarce numarul de elemente din vector